CSCI-1200 Data Structures
Test 1 — Practice Problems

Note: This packet contains selected practice problems from Test 1 from three previous years. Your test will contain approximately one third as many problems (totalling ∼100 pts). Problems may be extensions to specific homework assignments used that year.

1 Classy Grocery Bags [42 pts]

Ben Bitdiddle is building an app to streamline grocery shopping with his roommates. Here's an example of how they will write down the list of what they need to purchase:

```
std::vector<Food> shopping_list;
AddItem(shopping_list,"apple");       AddItem(shopping_list,"apple");
AddItem(shopping_list,"milk");        AddItem(shopping_list,"cereal");
AddItem(shopping_list,"eggs");        AddItem(shopping_list,"bread");
AddItem(shopping_list,"chicken");     AddItem(shopping_list,"cereal");
AddItem(shopping_list,"bread");       AddItem(shopping_list,"peanut_butter");
AddItem(shopping_list,"jam");         AddItem(shopping_list,"apple");
```

At the store, they will pack the groceries into bags, and Ben wants to make the bags equal in weight so no one gets stuck carrying too much on their way back to the apartment. To prepare, he looks up the typical weight of each item (a portion of the data file is shown below) and organizes the shopping list by weight.

```
LookupFoodWeights(shopping_list, "food_weights.txt");
std::sort(shopping_list.begin(), shopping_list.end(), heaviest_first);
for (unsigned int i = 0; i < shopping_list.size(); ++i) {
  shopping_list[i].print();
}
```

Here is the output of the above print statements for our example:

```
    milk 4.30
    chicken 2.50
    eggs 1.75
(2) bread 1.00
    peanut_butter 1.00
(2) cereal 0.75
(3) apple 0.33
    jam 0.25
```

food_weights.txt

```
eggs 1.75
jam 0.25
milk 4.3
bread 1.0
cereal 0.75
apple 0.33
peanut_butter 1.0
chicken 2.5
...
```

Ben has devised an algorithm below to place the items one-at-a-time into the bag that currently has the smallest total weight. In this example we split the groceries into 4 bags:

```
std::vector<Bag> bags(4);
for (unsigned int i = 0; i < shopping_list.size(); ++i) {
  for (unsigned int j = 0; j < shopping_list[i].getCount(); ++j) {
    bags[0].add(shopping_list[i].getName(), shopping_list[i].getWeight());
    std::sort(bags.begin(), bags.end(), lightest_first);
  }
}
for (unsigned int i = 0; i < bags.size(); ++i) {
  bags[i].print();
}
```

And here is the output of the print statements above for this example:

```
Bag { chicken cereal } total weight = 3.25
Bag { bread bread cereal apple jam } total weight = 3.33
Bag { eggs peanut_butter apple apple } total weight = 3.41
Bag { milk } total weight = 4.30
```

On the next 3 pages you will write the `Food` and `Bag` classes, including the helper functions for sorting, and write the `AddItem` and `LookupFoodWeights` helper functions. You should think about the implementation of the `print` functions and make sure your classes support their implementation, but you may skip writing the bodies of those functions.

1.1 `Food` Class Declaration [12 pts]

First, write the class declaration or *header file* (`food.h`) for the `Food` object. Use `const` and reference where appropriate. You may implement any one-line member functions in this file. Include any relevant helper functions for sorting. You may skip the `#includes` and include guards.

sample solution: 15 lines of code

1.2 `Food` Class Implementation [5 pts]

Now implement the remaining `Food` member and non-member functions as they would appear in the `food.cpp` file. You may skip the implementation of the `print` member function.

sample solution: 7 lines of code

1.3 `Bag` Class Declaration [6 pts]

Next, write the class declaration or *header file* (`bag.h`) for the `Bag` object. Use `const` and reference where appropriate. You may implement any one-line member functions in this file. Include any relevant helper functions for sorting. You may skip the `#include`s and include guards.

sample solution: 11 lines of code

1.4 `Bag` Class Implementation [6 pts]

Now implement the remaining `Bag` member and non-member functions as they would appear in the `bag.cpp` file. You may skip the implementation of the `print` member function.

sample solution: 7 lines of code

Finally, we can finish up the implementation by writing the last two necessary helper functions for this program. Think carefully about the use of `const` and reference.

1.5 Implement `AddItem` [6 pts]

sample solution: 8 lines of code

1.6 Implement `LookupFoodWeights` [7 pts]

sample solution: 13 lines of code

2 ASCII Art Kerning Analysis [25 pts]

For Homework 1 we printed every character with a fixed horizontal width of grid cells. The simple font file we used was 6 characters wide, and we always added a column of padding between characters, so each letter was placed an *offset* of 7 grid cells to the right of the previous letter. Professional font typography artists recognize that English letters are different widths and some pairs of letters can be placed more closely together, which allows us to print more characters per line of text without damaging legibility.

For example, with the simple font we used in Homework 1, the letter combination '`l`' + '`y`' can be printed with a *horizontal offset* of 3 grid cells for the second letter (below left). And the letter combination '`t`' + '`o`' can be printed with a *horizontal offset* of 5 grid cells for the second letter (below right).

```
#.....    ......    #........        ......    ......    ...........
#.....    #...#.    #..#...#.        .#....    .###..    .#....###..
#.....    #...#.    #..#...#.        ###...    #...#.    ###..#...#.
#.....    #...#.    #..#...#.        .#....    #...#.    .#...#...#.
#.....    .####.    #...####.        .#....    #...#.    .#...#...#.
.##...    ....#.    .##....#.        ..##..    .###..    ..##..###..
......    .###..    ....###..        ......    ......    ...........
```

Some other letter combinations, e.g., '`M`' + '`W`', must be printed with an offset of the full width + 1 = 6 + 1 = 7 grid cells. This offset is necessary to avoid overlap between the letters or having the letters touch each other and become illegible.

```
#....#    #....#    #....#.#....#
##..##    #....#    ##..##.#....#
#.##.#    #....#    #.##.#.#....#
#....#    #....#    #....#.#....#
#....#    #.##.#    #....#.#.##.#
#....#    .#..#.    #....#..#..#.
......    ......    ............
```

You will write a function named `KerningAnalysis` that takes in 5 arguments. The first two arguments are of type STL `vector` of STL `string`, which hold the ASCII art representations of the two characters from Homework 1. The third argument is the proposed integer `offset` for this letter pair. Your job is to detect if this offset is acceptable. The 4th and 5th arguments are booleans named `overlap` and `edge_or_diagonal`, which you will use to indicate if there is a problem with this offset. If the offset is *not sufficient*, your function should set one of the boolean variables to true to indicate that *either* portions of the letters will overlap *or* that the letters are too close to each other and they will touch on the edge of two grid squares or diagonally.

For example, analyzing an offset of 4 grid units for the letter combination '`+`' and '`L`' should detect an overlap (shown with the '`@`' symbol below). An offset of 5 grid units doesn't produce an overlap between the letters, but the letters do touch along a grid cell edge so the `edge_or_diagonal` would be set to true in this case. An offset of 6 is successful – it does not have any overlapping grid cells or any edge or diagonal grid cells touching between the two letters, so both booleans will be false.

```
......    #.....        ....#.....        .....#.....        ......#.....
..#...    #.....        ..#.#.....        ..#..#.....        ..#...#.....
..#...    #.....        ..#.#.....        ..#..#.....        ..#...#.....
#####.    #.....        ####@.....        ######.....        #####.#.....
..#...    #.....        ..#.#.....        ..#..#.....        ..#...#.....
..#...    ######        ..#.######        ..#..######        ..#...######
......    ......        ..........        ..........        ............
```

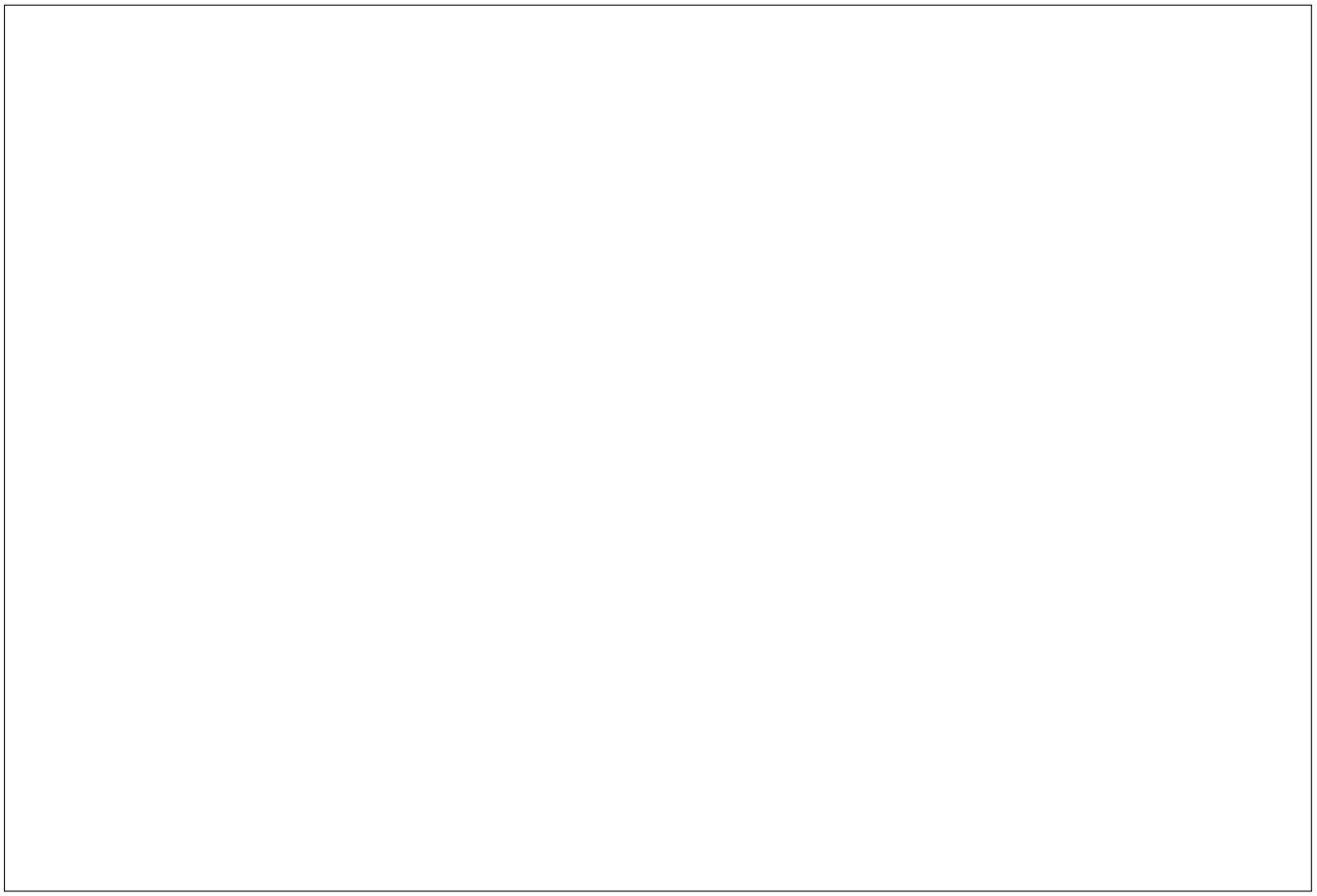
Write the `KerningAnalysis` function as described on the previous page. Be sure to use `const` and reference as appropriate.

sample solution: 28 lines of code

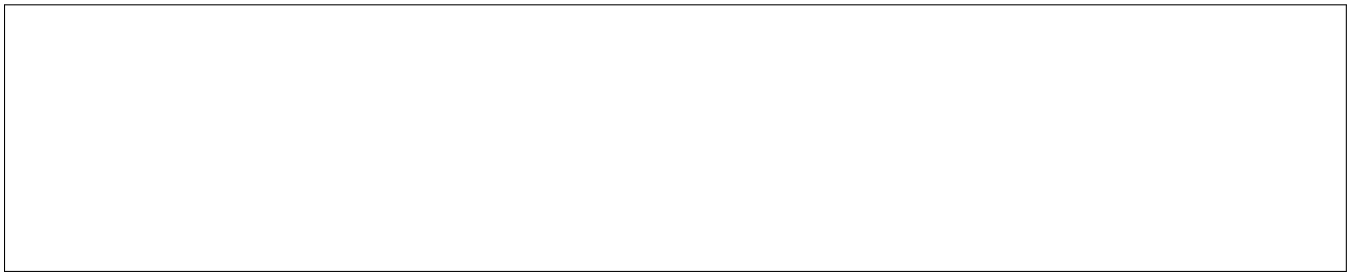
3 Diagramming Memory [15 pts]

In this problem you will work with pointers and dynamically allocated memory. The fragment of code below allocates and writes to memory on both *the stack* and *the heap*. Following the conventions from lecture, draw a picture of the memory after the execution of the statements below.

```
char* cat;
char** dog;
char fish[2];
char horse;
dog = new char*[3];
dog[0] = new char;
fish[0] = 'b';
fish[1] = 'i';
dog[1] = &fish[1];
dog[2] = &horse;
cat = dog[0];
*cat = 'r';
horse = 'd';
```



Now, write a fragment a C++ code that cleans up all dynamically allocated memory within the above example so that the program will not have a memory leak.



4 Literature & Math [15 pts]

We would like to compute basic statistics for each sentence in a classic children's story. An excerpt from the story is shown on the left and the corresponding output we produce is shown on the right.

```
A little steam engine had a long train of
cars to pull.  She went along very well
till she came to a steep hill. But then
no matter how hard she tried she could not
move the long train of cars.  She pulled and
she pulled. She puffed and she puffed. She
backed and started off again. Choo! Choo!
But no! The cars would not go up the hill.
```

continued

```
sentence  1 #words= 12 avg chars/word=  3.6
sentence  2 #words= 12 avg chars/word=  3.6
sentence  3 #words= 17 avg chars/word=  3.7
sentence  4 #words=  5 avg chars/word=  4.2
sentence  5 #words=  5 avg chars/word=  4.2
sentence  6 #words=  6 avg chars/word=  4.5
sentence  7 #words=  1 avg chars/word=  4.0
sentence  8 #words=  1 avg chars/word=  4.0
```

continued

You may assume that all sentences end with a period, exclamation point, or question mark. You may assume there is no other punctuation in the file. *NOTE: The placement of newlines within the input file is not relevant. Newlines are ignored.* You should include some formatting code to neatly present the output (the formatting syntax does not need to be perfect). Complete the program below:

```
int main() {
  std::ifstream istr("little_engine_that_could.txt");
  std::string word;
  int num_chars = 0;
  int num_words = 0;
  int num_sentences = 0;
  while (istr >> word) {
```

sample solution: 12 lines of code

```
  }
}
```

5 Carpet Warehouse Orders [35 pts]

Congratulations! You've just been offered a Summer Arch internship at 1-800-CARPETS! Here's a sample of customer orders for wall-to-wall carpet received on a typical day:

```
std::vector<Carpet> orders;
orders.push_back(Carpet(12, "white",   3.5));
orders.push_back(Carpet(15, "white",  19.5));
orders.push_back(Carpet(15, "blue",   10.0));
orders.push_back(Carpet(12, "silver", 10.5));
orders.push_back(Carpet(12, "white",   8.5));
orders.push_back(Carpet(15, "blue",   20.0));
```

The carpet is manufactured in large rolls in one of two widths: 12' or 15' (measured in feet), and is available in a wide variety of colors. The necessary length of carpet will be cut from one of these large rolls to fit the dimensions of the customer's room. The customer is charged based on the area (square feet).

After collecting and organizing the orders for the day, the sales staff print information about the carpet rolls that will need to be in stock at the warehouse to satisfy the customer orders. Note that orders for the same color and width have been combined, and that the table is neatly formatted.

```
   blue, 15' roll,  450.0 sq feet
 silver, 12' roll,  126.0 sq feet
  white, 12' roll,  144.0 sq feet
  white, 15' roll,  292.5 sq feet
```

5.1 `Carpet` Class Declaration [11 pts]

First, write the class declaration or *header file* (`carpet.h`) for the `Carpet` object. Use `const` and reference where appropriate. To demonstrate the C++ syntax necessary for more complex classes, please save the implementation of ALL functions (even one-liners) for the *class implementation* on the next page.

sample solution: 12 lines of code

5.2 `Carpet` Class Implementation [12 pts]

Next, implement all `Carpet` member functions (and any related non-member functions) as they would appear in the `carpet.cpp` file.

sample solution: 16 lines of code

5.3 Printing the Necessary Warehouse Stock [12 pts]

Finally, complete the code fragment below to print the carpet data in the `orders` variable used in our initial example. Closely study the desired sample output on the first page; however, your code should work for any data, not just the specific example shown above!

```
// organize the data

std::sort(                                                                    );


// initialize a few helper variables









// loop over the orders and print each necessary carpet roll
for (unsigned int i = 0; i < orders.size(); i++) {

















}
```

sample solution: 16 lines of code

In your answer above, you are encouraged to write and use a helper function named `PrintCarpetRoll` to format each carpet roll in the table using the STL `iomanip` library. Implement that function below:

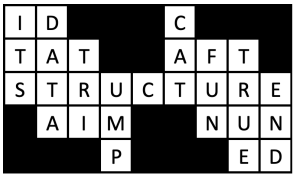
```


```

sample solution: 5 lines of code

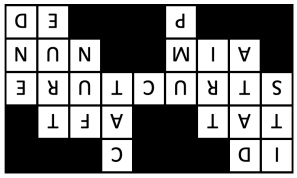
6 Analyzing Crossword Symmetries [20 pts]

It is a common convention for the black squares of a crossword puzzle to be *rotationally symmetric* - when spun around 180°, the shapes formed by the black squares will be in the same positions. Alternatively, puzzles may be *horizontally-mirrored* – black squares are in the same positions if flipped left to right.

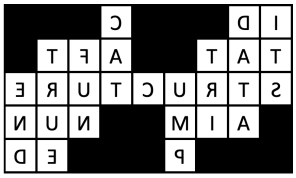
```
id###c###
tat##aft#
structure
#aim##nun
###p###ed
```



puzzle1

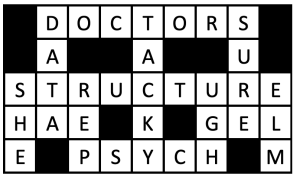


puzzle1 is *rotationally-symmetric*



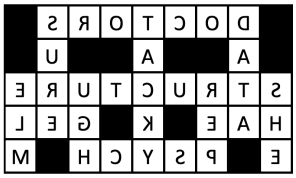
puzzle1 is *NOT horizontally-mirrored*

```
#doctors#
#a##a##u#
structure
had#k#gel
e#psych#m
```



puzzle2

puzzle2 is *NOT rotationally-symmetric*



puzzle2 is *horizontally-mirrored*

Write a function named `check_symmetries` that takes three arguments, an STL `vector` of STL `strings` named `board`, and two boolean variables: `rotationally-symmetric` and `horizontally-mirrored`. The function should analyze the symmetries of the board and set the boolean variables appropriately.

sample solution: 19 lines of code

7 `ben++`, the Better `clang++`? [24 pts]

Ben Bitdiddle has decided to write a better C++ compiler named `ben++`. He's asked you to write the compilation configuration parser, which has similar options and format to both `g++` and `clang++`, except that the configuration is read from a file, not the command line. Furthermore, Ben's compiler can be parallelized: if `-p` is specified, the next value is an integer number of threads for parallel computation.

Here are a few example valid compilation configuration files:

```
ben++ main.cpp date.cpp name.cpp -Wall -Wextra -p 4 -o lab_executable.out
```

```
ben++ main.cpp
```

```
ben++ -Wall date.cpp -o lab_executable.out name.cpp -Wextra -p 2 main.cpp
```

Ben would like his compiler to catch some common programmer mistakes. For example, parsing:

```
ben++ -Wall -Wextra -p 4 -o a.out
```

should print to `std::cerr` the message: "`ERROR: No source code files provided!`". And

```
ben++ main.cpp date.cpp date.h -Wall -Wextra -p 4 -o lab_executable.out
```

should result in this `std::cerr` message: "`WARNING: You should not directly compile a .h file!`".

7.1 More Error Checking is More Better! [4 pts]

There are many other compiler configuration typos or misunderstandings that a novice C++ programmer can experience when using `g++` or `clang++`. Have you or one of your Data Structures lab friends made compilation configuration mistakes? Give a specific example of another buggy compilation configuration. In 1-2 well-written sentences describe the mistake with your example. What descriptive error or warning message could be printed to `std::cerr` that would help a programmer fix this specific mistake?

7.2 Compilation Parsing Implementation [20 pts]

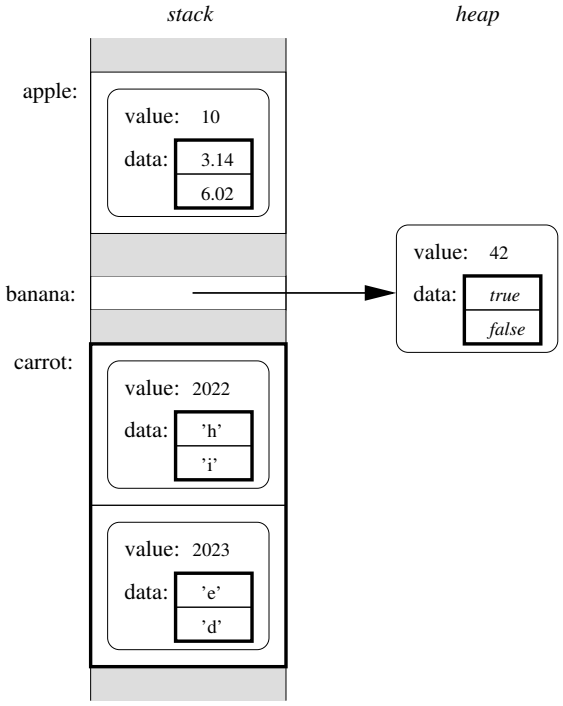
Write a fragment of code below to parse a compilation configuration file from the `istr` file stream variable. Once you've finished reading the file, call Ben's `DO_COMPILATION` function, which takes 4 arguments: an STL `vector` of STL `string`s that are the source code files, an STL `vector` of STL `string`s that are the compilation warnings to be enabled, the integer number of parallel threads (should default to 1), and an STL `string` indicating the executable filename (should default to `"a.out"`).

```
std::ifstream istr("compilation_configuration.txt");
```

sample solution: 27 lines of code

8 Object Memory [18]

The memory diagram below uses a class named `Object`. Write the class declaration and implement all member functions as they would appear in the `object.h` file.



sample solution: 15 lines of code

Finish the code fragment below using your `Object` class to create this memory diagram.

```
                                        apple;
apple.setValue(10);
apple.setData(3.14, 6.02);

                                        banana;
```

sample solution: 8 lines of code

9 Palindrome Art [15 pts]

Write a function named `palindrome` that will take in two arguments: the ASCII art message output from our Homework 1 as an STL `vector` of STL `string`s, and the integer width of a single letter (width=7 for this font). The function should return true if the message is a palindrome (a word containing the same letters when reversed), and false otherwise.

```
.............................................
.###....###....###....###....###....###....###...
#..........#..#......#...#..#..........#..#......
#.......####..#......####...#.......####..#......
#......#...#..#......#......#......#...#..#......
#.......####...###....####...###....####..#......
.............................................
```

```
.........................................
.###...####....####...###....####...###...
#...#..#...#..#...#......#..#...#..#...#..
####...#...#..#...#...####..#...#..####...
#......#...#...####..#...#...####..#......
.####..#...#......#...####......#...####..
...............###..........###..........
```

For example, the function should return true when given the top message. And false when given the bottom message. *Note: There is no kerning in this problem. (That was extra credit.)*

sample solution: 14 lines of code

10 Classic Trees [42 pts]

Over the pandemic, Louis B. Reasoner worked with RPI Environmental & Site Services to overhaul their system for tracking maintenance tasks on the arboreal specimens (a.k.a., trees) on campus. Here is an example of how the staff will interact with the tracking system. The following commands might appear in the `main` function. First, we can create a few specific trees and the year they were planted. The system allows us to schedule routine maintenance tasks for each tree.

```
Tree tallest("oak", 1950);
bool success = tallest.addMaintenance("fertilize");
assert (success == true);
Tree favorite("maple", 1995);
bool success = favorite.addMaintenance("water");
assert (success == true);
```

Next we use an STL `vector` container to store these and other trees:

```
std::vector<Tree> trees;
trees.push_back(favorite);
trees.push_back(tallest);
trees.push_back(Tree("pine",2002));
trees.push_back(Tree("oak",2012));
trees.push_back(Tree("ash",1990));
trees.push_back(Tree("maple",2015));
trees.push_back(Tree("maple",2008));
```

Below we schedule maintenance tasks to prune all trees that were planted in `2005` or earlier, to treat all `ash` trees for invasive insects, and to water all `maple` trees.

```
int prune_tasks = addMaintenance(trees,2005,"prune");
assert (prune_tasks == 4);
int insecticide_tasks = addMaintenance(trees,"ash","insecticide");
assert (insecticide_tasks == 1);
int water_tasks = addMaintenance(trees,"maple","water");
assert (water_tasks == 2);
```

Each of these function calls returns the number of trees that match and are newly scheduled to receive this maintenance. Note: We do not add duplicate maintenance tasks if a specific tree is already scheduled for that task.

Finally, we can print the data:

```
std::sort(trees.begin(),trees.end(),byNumTasks);
for (unsigned int i = 0; i < trees.size(); i++) {
  trees[i].print();
}
```

Which results in the following output, ordered first by number of scheduled tasks, the species, then age (planting year).

```
ash (1990) - prune insecticide
maple (1995) - water prune
oak (1950) - fertilize prune
maple (2015) - water
maple (2008) - water
pine (2002) - prune
oak (2012) -
```

10.1 `Tree` Class Declaration [16 pts]

Write the class declaration or *header file* (`tree.h`) for the `Tree` object. Use `const` and call by reference where appropriate. Remember, you should only implement 1 line member functions in the *class declaration*.

sample solution: 13 lines of code

10.2 `Tree` Class Member Function Implementation [10 pts]

Now implement the longer ($>$ 1 line) member functions as they would appear in the `tree.cpp` file.

sample solution: 14 lines of code

10.3 `Tree` Class Non Member Function Implementation [16 pts]

Finally, implement the three non-member functions used in the sample code related to the `Tree` class.

sample solution: 30 lines of code

11 Who's Eligible Now? [20 pts]

```
Smith 3
  Elizabeth 15.1
  Zachary 8.4
  Molly 2.1
Jones 1 Robert 12.2
Thomas 2
  Sally 2.7 John 4.9
```

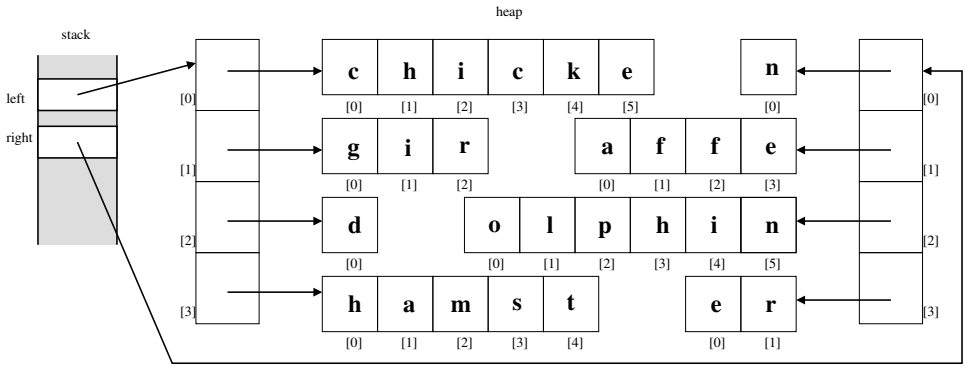
Anticipating the FDA's forthcoming approval Alyssa P. Hacker has been hired to prepare for the vaccination of children under 16 years of age. Help her write the `parse_children` function that takes in 3 arguments: the input filename of family data, the new minimum age for vaccination, and an initially-empty STL `vector` of `strings` to store the last names of families with at least one newly-eligible child (so we can send information about available vaccination appointments).

Each family in the file begins with the family's last name, then the number of children, and then each child's first name and floating point age. You should not use `getline` or `eof` to parse this file. The function returns the number of newly-eligible children (to buy sufficient doses). For this example, if the minimum age is 5.0, the `vector` should contain 2 `string`s (`Smith` and `Jones`) and the function should return 3.

sample solution: 21 lines of code

12 DynaMax Memory Cut [20]

We start with an STL `vector` of STL `strings` named `words`; for example: `chicken`, `giraffe`, `dolphin`, and `hamster`. Your task is to complete the code fragment below to construct the memory diagram on the right.



The diagram stores all of the characters from each word in arrays that are split or cut between the neighboring pair of letters that are furthest apart alphabetically.

`left =` ;

`right =` ;

```
for (unsigned int w = 0; w < words.size(); w++) {
    unsigned int split = 0;
    int max = 0;
    for (unsigned int j = 0; j < words[w].size()-1; j++) {
        int diff = abs(words[w][j]-words[w][j+1]);
        if (diff > max) { split = j; max = diff; }
    }
```

sample solution: 9 lines of code

```
}
```